Interactive out-of-core visualization of very large multiresolution time series scientific data

Final Report

R. Daniel Bergeron

August 19, 2010

1. Project goals

The principal final goals for the year as described in the year 2 progress report are shown below. These goals are the culmination goals for the entire project since they are all continuations of previous efforts.

Goal
1. Extend *VisIt/STARgen* interface
2. Multiresolution data generation by decimation and wavelet transformation
3. File compression research
4. Adaptive resolution prototype
5. *VisIt* data interface: AR support
6. Error model design/implementation
7. Final evaluation

2. Achievements

2.1. Extend VisIt/STARgen package

We identified 5 extensions to the VisIt/STAR interface that we hoped to implement:

1. more flexibility in the specification of resolution level (beyond 3 fixed levels);
2. support for automatic resolution change in response to viewing specification changes;
3. support for out-of-core rendering in VisIt;
4. support for VisIt access to Granite compressed storage functionality; and
5. support for access to adaptive resolution data.

In addition we submitted a version of our STARgen data plugin to the VisIt development group. That plugin was accepted and will be distributed as part of the next official VisIt release due out in Summer 2010.

2.1.1. More flexible resolution specification

Our initial implementation of the VisIt STARgen interface only supported 3 pre-defined resolution levels. We extended this to support an arbitrary number of resolution levels; the only requirement is that the different data resolutions must be created in a preprocessing step.

2.1.2. Automatic resolution change

Our multiresolution data support is an ideal foundation upon which to build an automatic resolution change: as the user zooms into the scene (by changing the viewing parameters), the system automatically accesses higher resolution data and zooming out would cause a switch to lower resolution data. We have implemented this effectively in standalone applications.

Incorporating this into our VisIt interface has proved to be infeasible. The VisIt environment supports user written *data* plugins, which is how we are able to integrate our data model into VisIt. Unfortunately, however, there is no mechanism that allows a *data* plugin to access the current viewing parameters or for the interactive viewing modules to communicate with a *data* plugin. We investigated the possibility of making local modifications to the VisIt core to provide this support, but were unable to identify a way of doing this that did not require many code modifications in many different modules. We decided to abandon this goal.

2.1.3. Out-of-core rendering using VisIt

Using the *domain* functionality of VisIt, we implemented a version of our VisIt data plugin that, in effect, allows VisIt to render data sets that are too large to fit in main memory. The data set is automatically partitioned into subsets that do fit in memory, and each subset is assigned to a domain. This functionality was added to VisIt to support parallel rendering, but it can also be used for out-of-core rendering on a single processor. Our implementation is complete and does achieve the goal of rendering a data set too large to fit in main memory. Unfortunately, however, the start up performance is very poor. Given a time series data set with 60 time steps, the system takes 15-20 minutes to complete a pass through the data. Once that first pass has finished, however, the subsequent passes complete in just a few minutes. The subsequent passes also require out-of-core rendering, so we believe that our fundamental architecture and implementation is appropriate. Even with help from members of the VisIt development group, we have not yet been able to identify the problem. Consequently, this version of our data plugin will not be released for general use until we are able to address the issue effectively.

2.1.4. Access to compressed Granite data sources

We had planned to implement a VisIt interface to an extension of the Granite system that was intended to achieve better I/O performance by doing block-oriented LWZ compression of large scientific data sets. The preliminary implementation of that system did not provide sufficient improvement to justify the overhead. We redirected our efforts in this area to explore a novel approach to wavelet decompression intended to achieve a similar result: less memory required to store the same resolution data with less error. This effort is described in more detail in section 2.3.

2.1.5. VisIt access to error-driven adaptive resolution data

Our prototype error-driven adaptive resolution software has been built on top of the *Silo* utility of *VisIt* (*Silo: A mesh and field I/O library and scientific database*. Lawrence Livermore National Laboratory, Livermore, CA. See http://wci.llnl.gov/codes/silo/index.html.) Although we have not yet completed a *VisIt* interactive plugin for this tool, users can use *VisIt* to browse through and render the various resolutions generated by our tool. Section 2.2 describes our tool.

2.1.6. VisIt approved data plugin

Our multiresolution data access plugin has been accepted by the VisIt development group and will be distributed with VisIt beginning with the next VisIt release due out in Summer 2010. This version of the Visit/STARgen interface only supports multiresolution data. This functionality is the easiest to support for a wide range of applications and data sets, and it only requires minimal setup using very general-purpose software. In the future, we do plan to submit a version of the plugin that supports the complete error-driven adaptive resolution data representation model described below.

2.2. Error-driven adaptive resolution

We have completed our prototype implementation of a software tool based on the expanded error model described in last year's report. The key innovation of the expanded error model is that the error calculations can be specified at lower resolutions than the corresponding data. This tool allows the scientist to create multiple fully-renderable error-driven adaptive resolutions from a single, high-resolution data set. The software supports the following features.

1. Lower resolution data can be generated using either uniform decimation or wavelet decomposition.
2. The user can specify one or more *error resolution level offsets*; the resolution offset is the difference between the resolution of the data and the resolution of the error. We have found that it is almost never necessary to maintain error information at the same resolution level as the data. In fact, in addition to taking much less space, lower resolution error allows us to utilize a much broader range of error measures that can provide more effective error information to the user. The most important requirement is that we provide *localized* error information, which allows a user to interactively zoom in to subregions of high error at the current resolution by accessing higher resolution data in those regions.
3. The user can specify one or more error measures from the choices: maximum absolute error, two versions of maximum relative error, average absolute error, two versions of average relative error, signal-to-noise ratio, and standard deviation.
4. New user-defined error measures can be implemented as Java subclasses.
5. The user can specify the maximum number of data resolution levels to generate. Typically, once the uniform resolution data representation is small enough to provide highly interactive browsing of the data, there is no need to create lower resolutions.
6. All the error calculations are done as a preprocessing step that generates the lower resolution data representations, along with error information stored in an octree-based data structure.
7. Error tolerances for defining the adaptive resolution access to the data are defined by the user at *run time*. The data access software browses through the error octree to identify and access the data at the appropriate (adaptive) resolution to satisfy the current error tolerance specification.

We evaluated the performance of these adaptive resolutions generated with various parameters compared to the original data set. We found that adaptive resolutions generated with reasonable subdomain sizes and error tolerances show significantly improved performance during visualization compared to the original data sets. The most dramatic improvements come with very large data sets that cannot fit in main memory. In these cases, the adaptive resolution representation often so significantly reduces the size of data that needs to be accessed that interactive browsing is possible.

Details of this software and a preliminary evaluation are described in a paper that has been submitted the VDA 2011 conference [5]. A more complete description with more evaluation is included in a master's thesis [6]. A journal paper version of this information is in preparation.

Figure 1 shows an adaptive resolution rendering of a slice of a time step from the OpenGGCM MHD simulation. The adaptive resolution was generated using an error tolerance of 10%; the left image shows the rendering with the adaptive resolution grid super-imposed over the image; the right image is the same rendering without the grid.

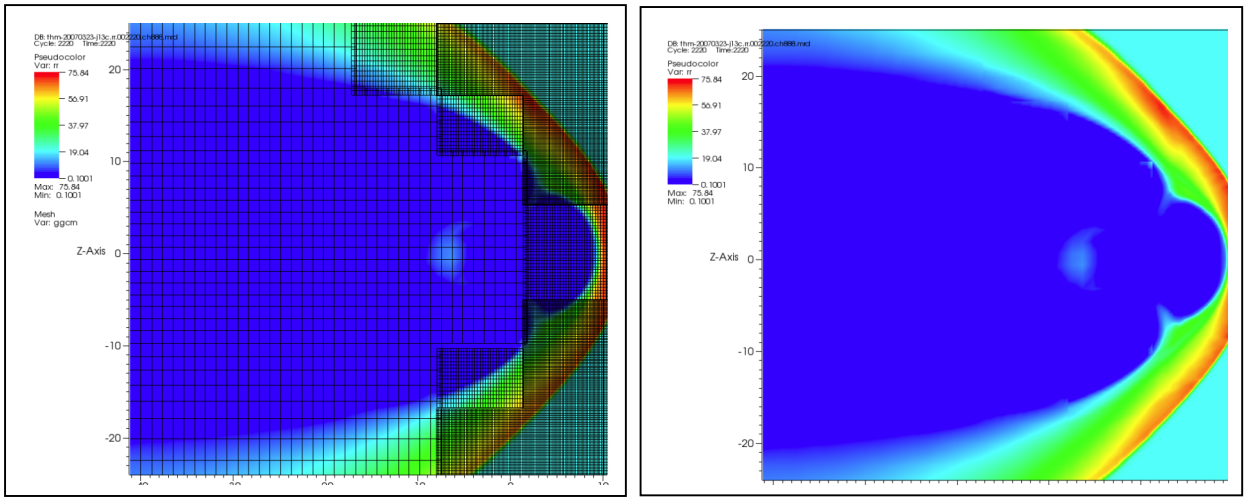

Figure 1. Rendering one slice of an adaptive resolution pseudocolor volume rendering of one time step of an MHD time series. The error tolerance is set to 10%. The adaptive mesh is shown on the left

Figure 2 shows the I/O and rendering times for the entire time series using the VisIt environment. Although the adaptive resolution times are significantly (and consistently) better than the larger uniform resolution rendering, these times are not interactive times. We can reach interactive time with higher error tolerances, and we are currently evaluating alternative interfaces in the VisIt environment that appear to be giving us significantly more efficient rendering times.

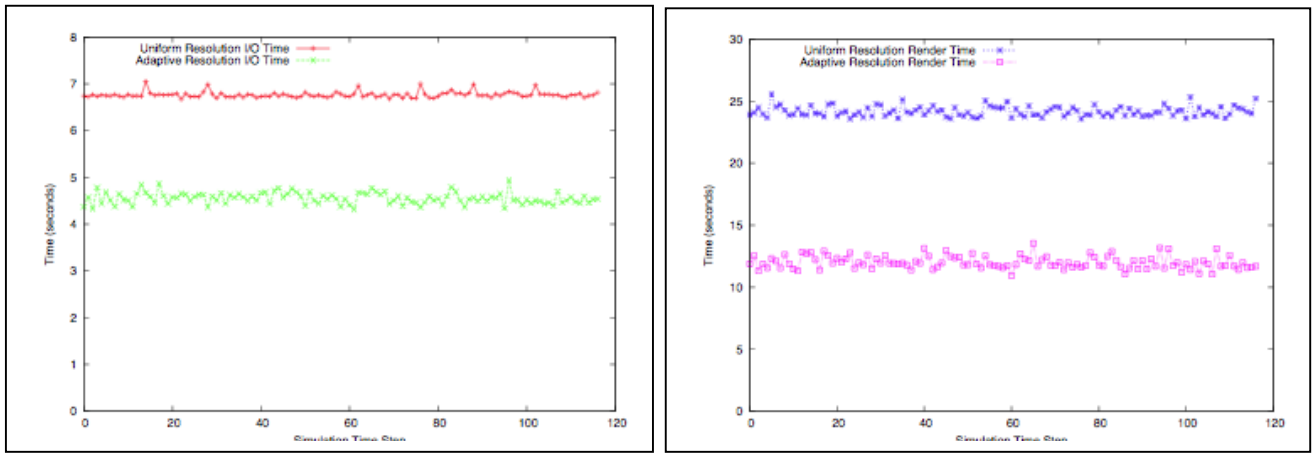

Figure 2. The left graph shows the I/O time per time step for the uniform resolution (upper red line) and the adaptive resolution representation using a 10% error tolerance (lower green line). The right graph shows the uniform resolution rendering time (upper blue) compared to the adaptive resolution rendering time (lower magenta).

2.3. Wavelet file compression research

A wavelet decomposition of a volume data set produces a *summary* component that is 1/8 the size of the original data and can serve as a very effective lower resolution representation of the original data set. The wavelet decomposition, however, also produces 7 sets of *detail* coefficients that can be used to reconstruct the original data with only minimal numerical error. There have been several algorithms proposed for saving some of the detail coefficients in order to reconstruct the original data with less error than would be the case by saving only the summary data. We have developed a new version of this idea, in which we pre-compute the error reduction that results from retaining each of the 7 sets of detail coefficients independently and keep only the *k* detail steps necessary to achieve the desired error/space criteria. In addition we have explored the efficacy of storing the detail coefficients as byte data, rather than float data.

This basic idea can be applied in three different contexts:

1. For the first application of the wavelet transform, we can use the summary data along with the *k* detail sets that are needed to achieve a specified error tolerance. An offline pre-computation step orders the detail blocks based on the error reduction that occurs by adding each block into the reconstruction step. During interaction, the current error tolerance is used to determine which detail blocks to load (or whether to move to the next higher resolution data).
2. In many cases of very large data sets, the summary output from a wavelet transform is a pretty good representation of the original data, but is still too large to fit in main memory. In this situation, we can apply a second wavelet transform to the summary data and save the summary of the summary and the *k* detail sets from the second transform that would satisfy a given error tolerance (as in case 1).
3. For datasets that represent reasonably continuous phenomena, the wavelet detail coefficients are typically much smaller in magnitude than the data values from which they are computed. Furthermore, the smaller the magnitude of the coefficient, the less important it is that its exact value be preserved. Consequently, we convert the wavelet coefficients from floating point values to a byte that is used as an index into a lookup table of values. The lookup table values are computed as a non-linear mapping between the index space and the space represented by the detail coefficients. In effect, we partition the detail coefficients into 256 bins in a way that allocates more bins for larger values than for smaller values. This means that we can allocate more bits of precision for the larger values, which are the ones that are most important to save.

For 3D data, the different options to represent the original data (or any summary component of the data at a lower resolution) are summarized in the table below as percentages of the space required to represent the data as floats. “S” identifies the summary component stored as floats, “Dk” represents the k best detail components represented as bytes.

Space savings from using bytes for detail coefficients as % of space used by floats **S+Dk represents using the summary plus the “best” k detail coefficient sets**								
Floats	**S**	**S+D1**	**S+D2**	**S+D3**	**S+D4**	**S+D5**	**S+D6**	**S+D7**
100	12.5	15.625	18.75	21.875	25	28.125	31.25	34.375

This approach provides the user with a continuum of space/error choices – as more space is used, the representation requires more I/O time, but includes less error. Furthermore, our preprocessing

step provides the user with error information so that the user can control the error/interactivity balance intelligently. Our preliminary experiments show that for large data sets the reduced I/O (or network transmission) time can make the difference between interactive and non-interactive data browsing. We are currently in the process of doing performance analysis on sample data sets to quantify more precisely the space/error/interactivity tradeoffs.

2.4. Error representation

We have implemented a prototype version of software to support our error model for multiresolution data. This tool generates accurate error information for each resolution of our data hierarchy when the hierarchy is generated using wavelet transformations. The error data has the same resolution as the data at that resolution and represents the approximate *local* error that occurs if this resolution is used to reconstruct the original data. We can generate both *maximum* and *average* error information; each are valuable for different kinds of user goals. The error is of the same format as the data and can be rendered by any *VisIt* rendering tool.

2.5. Final summary

2.5.1. Accomplishments

We have completed prototype versions of the major software tools that we initially proposed: multiresolution and adaptive resolution data representations driven by a variety of error models. Our performance evaluations show that significant I/O savings can be achieved at modest (and known) error levels.

We have created VisIt data plugins that support access to our MR and AR data representations as well as a VisIt rendering plugin that includes 4 different 3D flow rendering visualizations based particle-tracing. These renderings show unsteady flow fields and can be combined with other 3D VisIt visualization tools such as pseudocolor volume rendering and slice visualization. The VisIt data plugin for the MR data has been incorporated into the current VisIt distribution. We plan to submit the AR plugin in the near future.

2.5.2. Unachieved and under-achieved goals

We had originally hoped to incorporate our tools into a grid based environment. We eliminated that goal in year 2; it was clearly too ambitious.

Our principal goal with this research is to achieve interactive browsing of very large time series data sets by reducing the I/O bottleneck by trading off some error for significantly reduced I/O. Our prototype evaluations results are promising, but we're not there yet. We have been able to achieve interactive (2-3 fps) rendering of AR time series data sets that are so large that they thrash at their full resolution. However, we have not yet been able to show convincingly that the AR approach is consistently better than low uniform resolution browsing with higher uniform resolution zooming. We believe this is true, but we still have to add a few more features and improve our interface to VisIt (or find a different visualization environment).

VisIt has been both a boon and a bane. It is extremely valuable to be able to tap into an existing powerful and flexible visualization environment. On the other hand, it is an extremely large and complicated system with significant design and implementation limitations. Several features that we had planned to implement are simply not feasible to do within the VisIt framework, including the automatic interactive resolution change as the user zooms into and out of the data. It is also particularly difficult to figure out why some operations take as much time as they do. On a couple of occasions (including the out-of-core rendering application described in section 2.1.3), we implemented code that appears to be consistent with VisIt demos or existing applications and does implement the expected behavior, but does it much more slowly than expected. Most of our

efforts related to VisIt have taken much more code and much more debugging time than we had imagined.

2.5.3. Future plans

The major software tools produced as part of this grant are still under active development. Some of this continuing effort is being carried out by UNH graduate students who are completing theses or projects as part of their degree requirements. Some of the continuing work is now supported by an NSF grant.

3. Publications

All documents can be accessed from http://www.cs.unh.edu/star/research.htm

1. Bergeron, R.D. and A. Foulks, "Interactive Out-of-Core Visualization of Multiresolution Time Series Data", in *Numerical Modeling of Space Flows*: *1ST IGPP – CalSpace International Conference*, ed. Nikolai V. Pogorelov and Gary P. Zank, ASP Conference Series, Vol 359, Astronomical Society of the Pacific, 2006, pp. 285-294.
2. Foulks, A., D. Benedetto, R.D. Bergeron and T.M. Sparr, STARdata: A Data Server for Multiresolution Time Series Data, *AGU Fall Meeting Abstracts,* Dec. 2006, p. A1316+.
3. Foulks, A. and R.D. Bergeron, Multiresolution Data Access Within The VisIt Visualization Environment, Proceedings of the NASA Science Technology Conference 2007, June 19-21, University of Maryland University College. http://esto.nasa.gov/conferences/nstc2007/papers/Foulks_Andrew_A11P1_NSTC-07-0066.pdf
4. Foulks, A. and R.D. Bergeron, "Uncertainty visualization in the VisIt visualization environment", in *Proc. of SPIE Visualization and Data Analysis 2009*, SPIE Vol 7243, January 2009.
5. Foulks, A., R.D. Bergeron and S.H. Vohr, Visualization of Dynamic Adaptive Resolution Scientific Data, accepted for presentation at *SPIE Visualization and Data Analysis 2011*, January 2011.
6. Vohr, S.H., Error-driven adaptive resolutions for large scientific data sets, M.S. thesis, Department of Computer Science, University of New Hampshire, September 2010.
7. *STARgui User's Guide*, http://www.cs.unh.edu/star/release/UsersGuide.doc.